SDN 系統建立

學號：0711540 姓名：林峻賢

壹、 研究動機

聽完教授講解完 SDN 概念後，感覺這個概念對於每個網路世代更新時帶來的硬體更新可以降低很多成本，所以我認為這個將會是未來的趨勢，並且 SDN 也發展數十年，網路上有較為豐富的文獻以及資源可以進行研究，相對應的論壇也擴展的較成熟，所以想要藉此機會徹底認識 SDN 系統的建立方式。

貳、 研究方法

我將研究方法分成三個階段進行，分別是學習 Open vSwitch 如何操作、學習 mininet 建立虛擬網路環境、以 POX 編寫 controller 以自動控制 OVS 行為，詳細內容如以下分析。

一、 OVS

OVS 有兩種模式，分別是由 controller 控制行為以及傳統 switch 行為，前者屬於 SDN 中 switch 應有的行為，OVS 將被動的接收 controller 發布的指令以完成 MAC learning；而後者可以自主完成 MAC learning。這裡我們很明顯要設置成前者，並且在與 controller 斷線之際要維持 secure mode，意即仍然保持被動，不可以主動更新 flow table。

要先使用 sudo /usr/local/share/openvswitch/scripts/ovs-ctl --system-id=random start 以啟動 OVS 的 daemon。

二、 Mininet

Mininet 可以使用 API 快速建立網路拓樸，亦可透過 python 提供的 package 進行客製化的拓樸設計，以下簡單介紹。

1. API 建立網路拓樸：

透過以下指令

Sudo mn --switch ovs --controller remote --topo tree,depth=3,fanout=2

--switch ovs：使用 OVS 作為 switch；--controller remote：使用遠端機器作為 controller；--topo tree,depth=3,fanout=2：高度為 3 的樹狀拓樸，每個 switch 扇出 2 個 hosts。

2. Python 建立客製化拓樸：

透過名為 mininet 的 package 完成各種實作，詳細內容可以參照圖一。

```
from mininet.cli import CLI
from mininet.net import Mininet
from mininet.node import RemoteController
from mininet.term import makeTerm
```

(a) 引用 mininet 的 package。

```
net = Mininet(controller=RemoteController)
c0 = net.addController('c0',ip='127.0.0.1', port=6633)

s1 = net.addSwitch( 's1' )
s2 = net.addSwitch( 's2' )
```

(b) 建立 mininet 的 class，並加入 controller 以及 OVS。

```
h1 = net.addHost( 'h1', mac='00:00:00:00:00:01')
h2 = net.addHost( 'h2', mac='00:00:00:00:00:02')
h3 = net.addHost( 'h3', mac='00:00:00:00:00:03')
h4 = net.addHost( 'h4', mac='00:00:00:00:00:04')
```

(c) 加入 hosts。

```
net.addLink( s1, h1 )
net.addLink( s1, h2 )
net.addLink( s1, h3 )
net.addLink( s3, h4 )
```

(d) 加入實體連線。

圖一，使用 python 客製化虛擬網路環境。

三、 POX controller

設計 controller 的管道有很多，其中 POX 是一個相較簡潔、新手友善的管道，對於想要將簡單的實驗想法實現是一個適當的工具。可透過 python 中名為 pox 的 package 進行客製化的設計，並且可以實作 controller 對於 PacketIn、新增/刪除 OVS 的 flow table 等等的行為。

能使得 OVS 順利運作的關鍵是 controller 需要妥善處理 packetin 類型的封包。首先 controller 只要一偵測到該事件，會將封包的 source address 以及 switch input port 更新至既有的 port table。接著決定封包的去留，若是封包屬於 LLDP 或是 source address 是 OVS 列管的黑名單都會直接捨棄。完成過濾後若有 multicast 服務則會直接進行 flood；檢查封包資訊是否已存在於 port table，有的話則進行轉送，無則進行 flood。最終需要將已更新的 port table 給予 OVS 使其能轉送曾經見過的地址，詳細圖文見圖二。

```
self.macToPort[packet.src] = event.port # 1
```

(a) 將 PacketIn 封包資訊更新至 port table。

```
if packet.type == packet.LLDP_TYPE or packet.dst.isBridgeFiltered():
  drop() # 2a
  return
```

(b) 若是屬於 LLDP 類型或是黑名單者預先過濾。

```
if packet.dst.is_multicast:
  flood() # 3a
```

(c) 若要求 multicast 服務則直接進行 flood。

```
else:
  if packet.dst not in self.macToPort: # 4
    flood("Port for %s unknown -- flooding" % (packet.dst,)) # 4a
```

(d) 否則進入細部檢查，若 PacketIn 封包沒有 port table 紀錄則會做 flood。

```
if port == event.port: # 5
  # 5a
  log.warning("Same port for packet from %s → %s on %s.%s.  Drop."
      % (packet.src, packet.dst, dpid_to_str(event.dpid), port))
  drop(10)
  return
```

(e) 若輸入埠號與輸出埠號相同會丟棄封包一陣子以避免封包折返轉送。

```
msg = of.ofp_flow_mod()
msg.match = of.ofp_match.from_packet(packet, event.port)
msg.idle_timeout = 10
msg.hard_timeout = 30
msg.actions.append(of.ofp_action_output(port = port))
msg.data = event.ofp # 6a
self.connection.send(msg)
```

(f) 以更新的 port table 更新給 OVS。

圖二，controller 的設計邏輯。

參、 實驗環境

OS：Ubuntu 20.04 2GB MEM 20GB SSD

肆、 實驗結果

我將根據 mininet 提供的 API 指令以及以 python 自由設計網路拓樸的方式建立實驗環境，針對 API 指令共使用 tree、single 拓樸，各有兩種不同設計以測試基本網路拓樸在 controller 的控制之下使 OVS 達成轉送封包的能力；另外使用 python 建立比較複雜的網路拓樸以模擬現實世界可能會出現的景象，為了使拓樸複雜度有難易之分，將以 switch 有無環狀為出發點去設計。

基於電腦硬體限制，我的電腦在一台虛擬機運作之下有最佳的表現，所以我打算將 control plane 以及 data plane 一同實現在同個虛擬機上(儘管我的程式碼支援分離的，只需要將 remote IP 改成遠端電腦 IP 即可)。此外由於我的目的是希望建立 SDN 系統，將會把測試對象針對 controller 的 MAC learning 演算法是否能使得 hosts 之間能夠互 ping，至於路徑演算法就不會有過多著墨。而在所有實驗開始之前，要先將 controller (server)端的程式預先建立，以備未來 mininet 建立虛擬環境時 OVS 可以與 controller 接軌，詳細操作如圖三。完成 controller 建立之後，實驗環境將詳細由以下數點介紹：

```
james@ubuntu:~/pox$ ./pox.py forwarding.l2_learning

INFO:core:POX 0.2.0 (carp) is up.
```

(a) 開啟後進入待命階段。

```
james@ubuntu:~/pox$ ./pox.py forwarding.l2_learning

INFO:core:POX 0.2.0 (carp) is up.
INFO:openflow.of_01:[00-00-00-00-00-04 1] connected
INFO:openflow.of_01:[00-00-00-00-00-05 2] connected
INFO:openflow.of_01:[00-00-00-00-00-01 3] connected
INFO:openflow.of_01:[00-00-00-00-00-03 4] connected
INFO:openflow.of_01:[00-00-00-00-00-02 5] connected
INFO:openflow.of_01:[00-00-00-00-00-04 1] disconnected
INFO:openflow.of_01:[00-00-00-00-00-03 4] disconnected
INFO:openflow.of_01:[00-00-00-00-00-01 3] disconnected
INFO:openflow.of_01:[00-00-00-00-00-05 2] disconnected
INFO:openflow.of_01:[00-00-00-00-00-02 5] disconnected
```

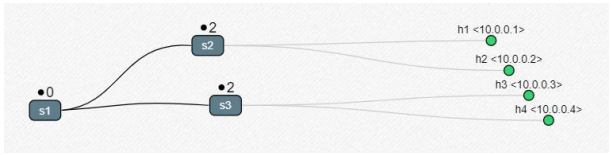
(b) 接管/斷線 OVS 時的資訊。

圖三，開啟新終端機並執行./pox.py forwarding.l2_learning 以啟動 controller 服務。

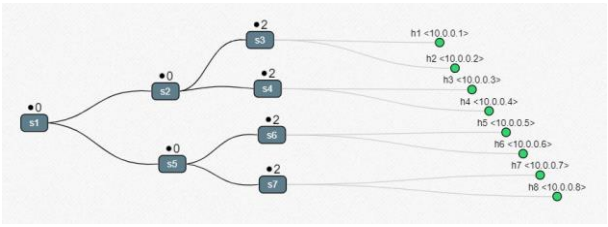
一、Tree topology

樹狀拓樸由 mininet 的 API 指令 sudo mn －topo tree,depth=X,fanout=Y 達成，X 表示樹的高度(深度)而每次節點要向外延伸 Y 個分支。如圖四所示，左側的(X,Y)=(2,2)，右側的(X,Y)=(3,2)，末端分別服務四位以及八位的 hosts，是一種最基礎的網路拓樸設計之一。

進入 mininet 的 CLI 介面可以下 pingall 指令，使兩兩 hosts 之間互相 ping，若成功即表示 OVS 的 flow table 有順利運作。如圖五所示，在這兩個基本拓樸中 controller 都可以順利的發會作用，可以看到最下面的一行說道 0% dropped 即表示沒有一個封包是失敗的。



(a) (X,Y)=(2,2)



(b) (X,Y)=(3,2)

圖四，基於 mininet 的 API 快速建立的 Tree topology，完整的指令為 sudo mn --switch ovs --controller remote --topo tree,depth=X,fanout=Y。

```
*** Ping: testing ping reachability
h1 -> h2 h3 h4
h2 -> h1 h3 h4
h3 -> h1 h2 h4
h4 -> h1 h2 h3
*** Results: 0% dropped (12/12 received)
```

(a) 共計 12 個互 ping 皆成功傳收。

```
*** Ping: testing ping reachability
h1 -> h2 h3 h4 h5 h6 h7 h8
h2 -> h1 h3 h4 h5 h6 h7 h8
h3 -> h1 h2 h4 h5 h6 h7 h8
h4 -> h1 h2 h3 h5 h6 h7 h8
h5 -> h1 h2 h3 h4 h6 h7 h8
h6 -> h1 h2 h3 h4 h5 h7 h8
h7 -> h1 h2 h3 h4 h5 h6 h8
h8 -> h1 h2 h3 h4 h5 h6 h7
*** Results: 0% dropped (56/56 received)
```

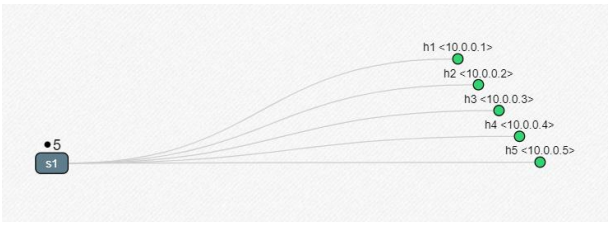
(b) 共計 56 個互 ping 皆成功傳收。

圖五，Tree topology 使用 pingall 指令的狀態。

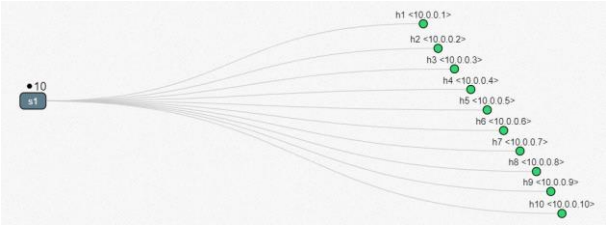
二、Single topology

單節點拓樸由 mininet 的 API 指令 sudo mn --switch ovs --controller remote --topo single,X 達成，X 表示 host 的數量。如圖六所示，左側的 X==5，右側的 X==10，末端分別服務五位以及十位的 hosts，是一種最基礎的網路拓樸設計之一。

進入 mininet 的 CLI 介面可以下 pingall 指令，使兩兩 hosts 之間互相 ping，若成功即表示 OVS 的 flow table 有順利運作。如圖七所示，在這兩個基本拓樸中 controller 都可以順利的發會作用，可以看到最下面的一行說道 0% dropped 即表示沒有一個封包是失敗的。



(a) X=5



(b) X=10

圖六，基於 mininet 的 API 快速建立的 Single topology，完整的指令為 sudo mn --switch ovs --controller remote --topo single,X。

```
*** Ping: testing ping reachability
h1 -> h2 h3 h4 h5
h2 -> h1 h3 h4 h5
h3 -> h1 h2 h4 h5
h4 -> h1 h2 h3 h5
h5 -> h1 h2 h3 h4
*** Results: 0% dropped (20/20 received)
```

(a) 共計 20 個互 ping 皆成功傳收。

```
*** Ping: testing ping reachability
h1 -> h2 h3 h4 h5 h6 h7 h8 h9 h10
h2 -> h1 h3 h4 h5 h6 h7 h8 h9 h10
h3 -> h1 h2 h4 h5 h6 h7 h8 h9 h10
h4 -> h1 h2 h3 h5 h6 h7 h8 h9 h10
h5 -> h1 h2 h3 h4 h6 h7 h8 h9 h10
h6 -> h1 h2 h3 h4 h5 h7 h8 h9 h10
h7 -> h1 h2 h3 h4 h5 h6 h8 h9 h10
h8 -> h1 h2 h3 h4 h5 h6 h7 h9 h10
h9 -> h1 h2 h3 h4 h5 h6 h7 h8 h10
h10 -> h1 h2 h3 h4 h5 h6 h7 h8 h9
*** Results: 0% dropped (90/90 received)
```

(b) 共計 90 個互 ping 皆成功傳收。

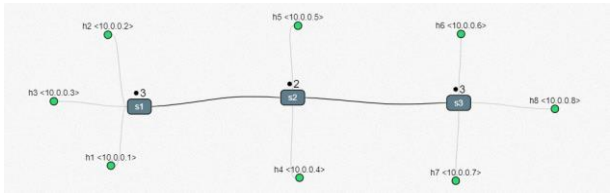
圖七，Single topology 使用 pingall 指令的狀態。

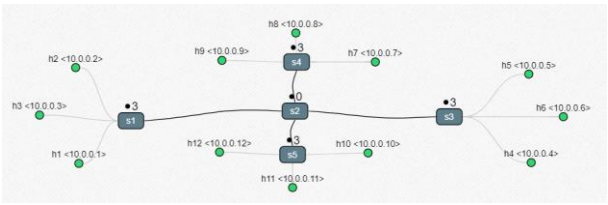
三、自製拓樸

1. 無環狀

自行設計的網路拓樸，設計概念是想要所有 OVS 都需要服務至少一台 host，並且每一個 OVS 都將接受同一個 controller 的指揮。詳細的設計圖如圖八，左側與右側分別設計了三個與五個 OVS，每個 OVS 都需要服務至少兩個 hosts，雖然深度只有一層，不過經由上述的兩個基礎實驗已經可以證實多層數的拓樸是可以成功運行的，因此這裡便不再多做測試。

進入 mininet 的 CLI 介面可以下 pingall 指令，使兩兩 hosts 之間互相 ping，若成功即表示 OVS 的 flow table 有順利運作。如圖九所示，在這兩個進階拓樸中 controller 都可以順利的發會作用，可以看到最下面的一行說道 0% dropped 即表示沒有一個封包是失敗的。





(a) 三個 OVS 以及八個 hosts。　　(b) 五個 hosts 以及十二個 hosts。

圖八，使用 python 自行設計的網路拓樸，主要特色是每個 OVS 至少需要服務兩台的 hosts。

```
*** Ping: testing ping reachability
h1 -> h2 h3 h4 h5 h6 h7 h8
h2 -> h1 h3 h4 h5 h6 h7 h8
h3 -> h1 h2 h4 h5 h6 h7 h8
h4 -> h1 h2 h3 h5 h6 h7 h8
h5 -> h1 h2 h3 h4 h6 h7 h8
h6 -> h1 h2 h3 h4 h5 h7 h8
h7 -> h1 h2 h3 h4 h5 h6 h8
h8 -> h1 h2 h3 h4 h5 h6 h7
*** Results: 0% dropped (56/56 received)
```

```
*** Ping: testing ping reachability
h1 -> h2 h3 h4 h5 h6 h7 h8 h9 h10 h11 h12
h2 -> h1 h3 h4 h5 h6 h7 h8 h9 h10 h11 h12
h3 -> h1 h2 h4 h5 h6 h7 h8 h9 h10 h11 h12
h4 -> h1 h2 h3 h5 h6 h7 h8 h9 h10 h11 h12
h5 -> h1 h2 h3 h4 h6 h7 h8 h9 h10 h11 h12
h6 -> h1 h2 h3 h4 h5 h7 h8 h9 h10 h11 h12
h7 -> h1 h2 h3 h4 h5 h6 h8 h9 h10 h11 h12
h8 -> h1 h2 h3 h4 h5 h6 h7 h9 h10 h11 h12
h9 -> h1 h2 h3 h4 h5 h6 h7 h8 h10 h11 h12
h10 -> h1 h2 h3 h4 h5 h6 h7 h8 h9 h11 h12
h11 -> h1 h2 h3 h4 h5 h6 h7 h8 h9 h10 h12
h12 -> h1 h2 h3 h4 h5 h6 h7 h8 h9 h10 h11
*** Results: 0% dropped (132/132 received)
```

(a) 共計 56 個互 ping 皆成功傳收。　　(b) 共計 132 個互 ping 皆成功傳收。

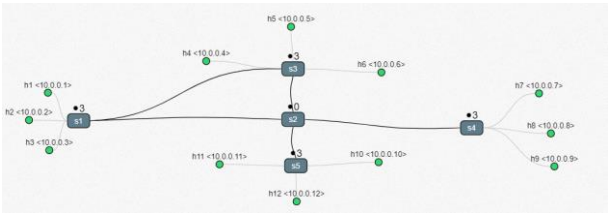
圖九，自製無環狀拓樸使用 pingall 指令的狀態。

2. 有環狀

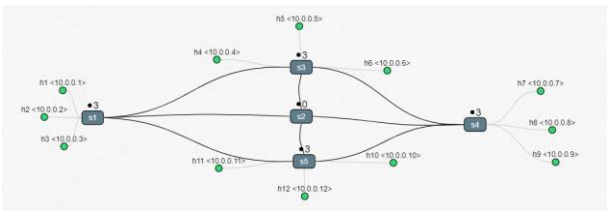
在現實的網路拓樸中最重要的因素之一的就是 redundancy。上述的拓樸設計中，若其中一台 OVS 發生損壞將會使得一定數量的 hosts 與整體網路失去聯繫，而最簡單的解方之一便是盡可能稠密化 OVS 間的實體連線，如此若還是發生 OVS 損壞問題，便可透過其他路徑找到備援 OVS 提供服務。

自行設計的網路拓樸，設計概念是想要所有 OVS 都需要服務至少一台 host，並且每一個 OVS 都將接受同一個 controller 的指揮之外，還要刻意產生迴圈。詳細的設計圖如圖十，左側與右側都使用了五個 OVS，不過分別設計有一個以及四個迴圈。

進入 mininet 的 CLI 介面可以下 pingall 指令，使兩兩 hosts 之間互相 ping，若成功即表示 OVS 的 flow table 有順利運作。如圖十一所示，在這兩個進階拓樸中 controller 都可以順利的發會作用，可以看到最下面的一行說道 0% dropped 即表示沒有一個封包是失敗的。



(a) 五個 OVS 以及十二個 hosts，並且帶有一個迴圈。



(b) 五個 hosts 以及十二個 hosts，並且帶有四個迴圈。

圖十，使用 python 自行設計的網路拓樸，主要特色是拓樸包含環狀設計。

```
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 h3 h4 h5 h6 h7 h8 h9 h10 h11 h12
h2 -> h1 h3 h4 h5 h6 h7 h8 h9 h10 h11 h12
h3 -> h1 h2 h4 h5 h6 h7 h8 h9 h10 h11 h12
h4 -> h1 h2 h3 h5 h6 h7 h8 h9 h10 h11 h12
h5 -> h1 h2 h3 h4 h6 h7 h8 h9 h10 h11 h12
h6 -> h1 h2 h3 h4 h5 h7 h8 h9 h10 h11 h12
h7 -> h1 h2 h3 h4 h5 h6 h8 h9 h10 h11 h12
h8 -> h1 h2 h3 h4 h5 h6 h7 h9 h10 h11 h12
h9 -> h1 h2 h3 h4 h5 h6 h7 h8 h10 h11 h12
h10 -> h1 h2 h3 h4 h5 h6 h7 h8 h9 h11 h12
h11 -> h1 h2 h3 h4 h5 h6 h7 h8 h9 h10 h12
h12 -> h1 h2 h3 h4 h5 h6 h7 h8 h9 h10 h11
*** Results: 0% dropped (132/132 received)
```

(a) 共計 132 個互 ping 皆成功傳收。

```
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 h3 h4 h5 h6 h7 h8 h9 h10 h11 h12
h2 -> h1 h3 h4 h5 h6 h7 h8 h9 h10 h11 h12
h3 -> h1 h2 h4 h5 h6 h7 h8 h9 h10 h11 h12
h4 -> h1 h2 h3 h5 h6 h7 h8 h9 h10 h11 h12
h5 -> h1 h2 h3 h4 h6 h7 h8 h9 h10 h11 h12
h6 -> h1 h2 h3 h4 h5 h7 h8 h9 h10 h11 h12
h7 -> h1 h2 h3 h4 h5 h6 h8 h9 h10 h11 h12
h8 -> h1 h2 h3 h4 h5 h6 h7 h9 h10 h11 h12
h9 -> h1 h2 h3 h4 h5 h6 h7 h8 h10 h11 h12
h10 -> h1 h2 h3 h4 h5 h6 h7 h8 h9 h11 h12
h11 -> h1 h2 h3 h4 h5 h6 h7 h8 h9 h10 h12
h12 -> h1 h2 h3 h4 h5 h6 h7 h8 h9 h10 h11
*** Results: 0% dropped (132/132 received)
```

(b) 共計 132 個互 ping 皆成功傳收。

圖十一，自製有環狀拓樸使用 pingall 指令的狀態。

伍、 結論

我認為以 SDN 系統建立作為題目是一個很好的期末專題，因為上課聽完教授講課後仍然對於 SDN 有一點距離感，或許是因為 SDN 在除了 controller 屬於技術面外，其餘部分則是偏向系統架構的重新設計，而課堂上講授的系統架構依稀只能大致理解而無法有深刻的代入感。而我認為最能破除這種距離感的關鍵在於真正理解並實作出 controller 的過程，儘管是使用比較簡單的 POX 仍可看出 controller 的原先設計是想要與 data plane 分離的，並且透過建立多個 OVS、hosts 進行實時操作並檢測，就能很清楚的看見 controller 是否有正確的規劃網路拓樸。整個實驗中，我遇到最大的困難在於處理環狀拓樸時的 controller 邏輯，原先的 OVS 的 flow table 在遇到重複的 MAC address 會發生錯誤，經過反覆的調整、驗證後才確認問題點並解決。總的來說，這次期末專題讓我更了解 SDN 以及轉送封包的細節。

陸、 參考文獻

[1] SDN 教學(一)

[2] SDN 教學(二)

[3] Mininet 視覺化工具